Adabas Calling Procedure

This chapter covers the following topics:

- Specifying an Adabas Call
- Syntax Conventions
- Control Block
- Format and Record Buffers
- Format Buffer Syntax
- Format Buffer Performance Considerations
- Record Buffer
- Format and Record Buffer Examples
- Search and Value Buffers
- Search Buffer Syntax
- Value Buffer
- Search/Value Buffer Examples
- ISN Buffer

Specifying an Adabas Call

This section describes the procedure used to call Adabas to execute an Adabas command; the syntax conventions used in the documentation; and the control block and buffers (format, record, search, value, and ISN) used to specify the Adabas command, and any additional information (parameters or operands) required for the command.

Adabas calls use the standard calling procedure provided by the host language (Assembler, COBOL, FORTRAN, or PL/I.

Note:
Examples of Adabas calls in each host language are provided with the programming examples in section *Programming Examples* of this documentation.

Each Adabas call must be accompanied by a parameter list. Each entry in the parameter list refers to a specific data area (buffer) defined in the user program. These buffers are used to transfer information to and from Adabas.

Parameter list entries must be provided in positional sequence. Parameters that are not used for a given command may be omitted provided that no required parameter follows; if it does, a dummy parameter must be provided.

The example below illustrates an Adabas call in COBOL.

```
CALL 'ADABAS' USING parameter-list
```

where "parameter-list" is a positional list of user-defined fields in the control block and user-defined buffers specified in the following order:

control-block fields	command code command ID file number response code ISN
buffers	format (fields to be read, updated) record (values returned, updated) search (search criteria) value (search values) ISN (ISNs resulting from search)

The Adabas calling sequence may be illustrated as follows:

```
user program
                .
                .
CALL 'ADABAS' USING parameter-list
                .
                .
CALL 'ADABAS' USING parameter-list
                .
                .
```

Syntax Conventions

This section describes the syntax conventions common to all user-defined data areas (buffers). Notation specific to a particular buffer is introduced in the discussion of that area later in this section.

Variables and Constants

A *variable* is indicated in lowercase, non-bold type:

```
name
```

In the actual entry, a valid value must be supplied for the variable.

A *constant* is indicated in uppercase type:

```
N
```

You must enter a constant exactly as shown in the syntax statement. Symbols (except for brackets, braces, vertical bars, underlines, and three consecutive periods, which are notational devices only) are treated as constants and must also be entered exactly as shown in the syntax.

Optional Elements, Optional Values, and Default Values

Elements enclosed in *brackets* ([ ]) are optional:

```
name [i]
```

indicates that a value for "name" is required and may occur by itself or with "i".

Default values may be in effect for optional elements when no value is specified. Such values are *underlined* (__).

The set of options or possible values for an element is enclosed in brackets or braces:

- Within *brackets* ([ ]), one of the options or values can (but need not) be specified;
- Within *braces* ({ }), one of the options or values *must* be specified; there is no default.

Vertical bars (|) may be used to separate options or possible values listed horizontally within brackets or braces. Options or possible values may also be indicated by stacking them vertically within brackets or braces without the separating vertical bars.

Repeating Elements

Three consecutive periods indicate that an element can be repeated:

```
format,   ...
```

indicates that more than one format can be entered when separated from the previous one by a comma. When *braces* ({ }) precede the repetition indicator, they delimit the elements that must be specified once, but may, as a group, be repeated (up to a maximum number of times normally specified in the text). For example:

```
{, name1 , name2 }...
```

indicates that the group ",name 1, name 2" must be specified once but may be repeated.

Example:

Syntax Example

This example is a selected portion of the record format part of the format buffer syntax. It is used here only to illustrate common syntax conventions. (See *Format Buffer Syntax, record-format* for information specific to the format buffer.)

Following is information about reading the example syntax:

- The "field-name" and a final period (.) elements must be specified (required). All other elements in the syntax statement are optional.
- If specified, "i" is concatenated with the "field-name" since there is no intervening delimiter.
- If "i" is specified, then either "C" or the optional specifications stacked below it can be specified, but not both.
- The optional specifications below "C" allow the following combinations: i, i(i), i(i-j), i-j, i-j(i), i-j(i-j).
- If specified, "length" and "data-format" must each be preceded by a comma.
- The term enclosed in braces ({ }) and followed by a comma may be repeated any number of times.

Control Block

The control block and the related buffers specify which Adabas command is to be executed and provide any additional information (parameters or operands) required for the command. The name of the control block must always be the first operand specified in an Adabas call. The control block itself must have the format shown in the following *Adabas Control Block Definition* table.

Field	Position Within Control Block	Length in Bytes	Format
(see discussion)	1	1	binary
(reserved)	2	1	binary
COMMAND CODE	3-4	2	alphanumeric
COMMAND ID	5-8	4	alphanumeric / binary
FILE NUMBER	9-10	2	binary
RESPONSE CODE	11-12	2	binary
ISN	13-16	4	binary
ISN LOWER LIMIT	17-20	4	binary
ISN QUANTITY	21-24	4	binary
FORMAT BUFFER LENGTH	25-26	2	binary
RECORD BUFFER LENGTH	27-28	2	binary
SEARCH BUFFER LENGTH	29-30	2	binary
VALUE BUFFER LENGTH	31-32	2	binary
ISN BUFFER LENGTH	33-34	2	binary

Field	Position Within Control Block	Length in Bytes	Format
COMMAND OPTION 1	35	1	alphanumeric
COMMAND OPTION 2	36	1	alphanumeric
ADDITIONS 1	37-44	8	alphanumeric / binary
ADDITIONS 2	45-48	4	alphanumeric / binary
ADDITIONS 3	49-56	8	alphanumeric
ADDITIONS 4	57-64	8	alphanumeric
ADDITIONS 5	65-72	8	alphanumeric / binary
COMMAND TIME	73-76	4	binary
USER AREA	77-80	4	not applicable

The use of each applicable field in the control block is explained with each Adabas command description in this documentation. To ensure user program compatibility with later Adabas releases, all control block fields not used by a particular command should be set to zeros or blanks, depending on field type as shown in the *Adabas Control Block Definition* table.

The position of each field in the control block is fixed. For example, the command option 1 field must always be in position 35.

All values in the control block must be entered in the data type defined for the field. For example, the ISN field is defined as binary format; therefore, any entry made in this field must be in binary format.

Notes:

1. Adabas and other Software AG program products use some control block fields for internal purposes, and may return values in some fields that have no meaning to the user. These uses and values may be release-dependent, and are not appropriate for program use. Software AG therefore recommends that you use only the fields and values described in this documentation. *In addition, you should always initialize unused control block fields with either zeros or blanks, according to their field types.*
2. Some Adabas-dependent Software AG products return control block values such as response codes and subcodes. Refer to the documentation for those products for a description of the product-specific control block values.

Control Block Fields

The content of the control block fields and buffers must be set before an Adabas command (call) is issued. Adabas also returns one or more values or codes in certain fields and buffers after each command is executed.

The following descriptions of the Adabas control block fields are valid for most Adabas commands; however, some Adabas commands use some control block fields for purposes other than those described here.

This section covers the following topics:

- Byte 1
- Command Code
- Command ID
- File Number
- Response Code
- ISN
- ISN Lower Limit
- ISN Quantity
- Buffer Length: Format, Record, Search, Value, and ISN
- Command Options 1/2
- Additions 1
- Additions 2
- Additions 3
- Additions 4
- Additions 5
- Command Time
- User Area

Byte 1

The first byte of the Adabas control block (ADACB) is used by the Adabas API to determine the processing to be performed. See *Linking Applications to Adabas* in the *Adabas Operations* documentation for more information.

The values for logical requests are:

Hex	Indicates ...
X’00’	a 1-byte file number (file numbers between 1 and 255) or DBID.
X’30’	a 2-byte file number (file numbers between 1 and 65535) or DBID.
X’40’	values greater than or equal to a blank. These are accepted as "logical application calls" to maintain compatibility with earlier releases of Adabas.

Note:
The X’44’, X’48’, and X’4C’ calls are reserved for use by Software AG and are therefore *not* accepted.

All other values in the first byte of the ADACB are reserved for use by Software AG.

Because an application can reset the value in the first byte of the ADACB on each call, it is possible to mix both one- and two-byte file number (DBID) requests in a single application. In this case, you must ensure the proper construction of the file number (ACBFNR) and response code (ACBRESP) fields in the ADACB for each call type. See the discussions of these fields for more information.

Software AG recommends that an application written to use two-byte file numbers always place X’30’ in the first byte of the ADACB, the logical database ID in the ACBRESP field, and the file number in the ACBFNR field. The application can then treat both the database ID and file number as 2-byte binary integers, regardless of the value for the file number in use.

Command Code

The command code defines the command to be executed, and comprises two alphanumeric characters (for example, OP, A1, BT).

Command ID

The command ID field is used by virtually all Adabas commands to identify users, their transactions, and decoded formats for reuse by subsequent instructions. The alphanumeric command ID is either "user-specified" or "system-generated", and can be either "local" for a given user’s internal formats or "global", allowing many users to access the decoded formats. See the section *General Programming Considerations* for more information about command IDs. For ET, CL, and some OP commands, Adabas returns a binary transaction sequence number in the command ID field.

File Number

Note:
For commands that operate on a coupled file pair, this field specifies the primary file from which ISNs or data are returned.

The file number may be one or two bytes.

Single-byte File Numbers and DBIDs

For an application program issuing Adabas commands for file numbers between 1 and 255 (single byte), build the control block as follows:

Position	Action
1	Place X’00’ in the first byte of the ADACB.
9	Place the file number in the second (rightmost) byte of the ACBFNR field of the ADACB. The first (leftmost) byte of the ACBFNR field is used to store the logical (database) ID or number.

If the first byte in ACBFNR is set to zero (B’0000 0000’), the Adabas API uses either the database ID from the ADARUN cards provided in DDCARD input data, or the default database ID value assembled into the link routine at offset X’80’. Applications written in Software AG’s Natural language need not include the first byte of the ADACB because Natural supplies appropriate values.

Double-byte File Numbers and DBIDs

Adabas permits the use of file numbers greater than 255 on logical requests. For an application program issuing Adabas commands for file numbers between 256 and 5000 (two-byte), build the control block as follows:

Position	Action
1	Place X’30’ in the first byte of the ADACB.
9	Use both bytes in ACBFNR for the file number, and use the two bytes in ACBRESP for the database (logical) ID.

If the ACBRESP field is zero, the Adabas API uses either the database ID from the ADARUN cards provided in DDCARD input data, or the default database ID value assembled into the link routine at offset X’80’.

Response Code

The response code field is used for two-byte database IDs.

It is also always set to a value when the Adabas command is completed. Successful completion is normally indicated by a response code of zero. For repeatable commands that process sequences of records or ISNs, other response codes indicate end-of-file or end-of-ISN-list. Non-zero response codes are defined in the *Adabas Messages and Codes*.

ISN

The ISN field both specifies a required four-byte Adabas ISN value required by the command and, where appropriate, returns either the first ISN of a command-generated ISN list, or an ISN of the record read by the command.

ISN Lower Limit

ISN lower limit specifies the starting point in an ISN list or range where processing is to begin. For OP commands, an optional user-specific non-activity timeout value can be specified in this field. An OP command also returns Adabas release information in this field (see also the additions 5 field description). When using the multifetch option, this field holds an optional maximum count of prefetched records to return; if zero, there is no limit.

ISN Quantity

The ISN quantity is a count of ISNs returned by a command. The count can be a total of all ISNs in an ISN list, or the total ISNs entered into the ISN buffer from a larger pool of ISNs by this operation. The OP command uses this field to specify an optional user-specific transaction time limit; it returns system and call type information flags in the ISN quantity field (see also the additions 5 field description). In addition, Sx commands using security-by-value set this field to 1 when *more than one* ISN meet the search criteria.

Buffer Length: Format, Record, Search, Value, and ISN

The format, record, search, value, and ISN buffer length fields specify the size of the related buffers. A buffer’s size usually remains the same throughout a transaction. In some ISN-related operations, the ISN buffer size value determines how a command processes ISNs; for example, specifying a zero ISN buffer length causes some commands to store a resulting ISN list in the Adabas work area. If a buffer is not needed for an Adabas command, the corresponding length value should be set to zero. In some cases (multifetch option, as an example), there is a limit on the length of the buffer; see the specific command descriptions for more information.

Command Options 1/2

The command option 1/2 fields allow you to specify processing options (ISN hold, command-level prefetching control, returning of ISNs, and so on).

Additions 1

The additions 1 field sometimes requires miscellaneous command-related parameters such as qualifying descriptors for creating ISN lists, or the second file number of a coupled file pair.

Additions 2

The additions 2 field returns compressed record length in the leftmost (high-order) two bytes and decompressed length of record buffer-selected fields in the rightmost (low-order) two bytes for all An, Ln, Nn, and S1/2/4 commands. OP (open) and RE (read ET data) commands return transaction sequence numbers in this field. If Entire Net-work is installed, some response codes return the node ID of the "problem" node in the leftmost two bytes of the additions 2 field.

If a command results in a nucleus response code, the addition 2 field's low-order (rightmost) two bytes (47 and 48) can contain a hexadecimal subcode to identify the cause of the response code. For example, if no OP command began the session and the ADARUN statement specified OPENRQ=YES, a response code 9, subcode 66 is returned and these bytes are set to the hexadecimal value 0042 corresponding to the decimal 66. Response codes and their subcodes (as decimal equivalents) are described in the section *Nucleus Response Codes* in the *Adabas Messages and Codes*.

Additions 3

The additions 3 field is for providing a user‘s password for accessing password-protected files. If the file containing the field is actually password-protected, the password in this field is replaced with spaces (blanks) during command execution before Adabas returns control to the user program.

Note:
Some proprietary/optional features such as the Adabas External Security Interface (ADAESI) set this password, and it therefore should not be overwritten.

Additions 4

The additions 4 field must be set to a cipher code for those instructions that read or write encrypted (ciphered) database data files. For commands requiring multiple command IDs but no cipher code, one of the command IDs is specified in this field.

When processed by the nucleus, an Adabas call returns the Adabas release (version and revision) level numbers and the database ID in the low-order (rightmost) *three* bytes of the additions 4 field with the format "vrnnnn" where

v	is the Adabas version number;
r	is the Adabas revision level number; and
nnnn	is the number (hexadecimal) of the Adabas database that processed the call.

For example, " 741111" indicates that an Adabas version 7.4 nucleus on database 4369 processed the call.

There is one exception to this format: If the call is processed by an Adabas nucleus lower than version 5, the value returned is "404000".

Additions 5

The high-order (leftmost) two bits of the first byte of the additions 5 field control the unique or global format ID selection; the low-order (rightmost) four or eight bytes can contain either an optional unique or global format ID, respectively. Refer to the section *Using a Global Format ID* for a complete description of this feature. A global format ID to be deleted can be specified in this field for the RC (release command ID) command. When completed, the OP command returns any optionally specified non-activity and/or transaction timeout values in the additions 5 field.

Command Time

The command time field is used by Adabas to return the elapsed time which was needed by the nucleus to process the command. This does not include the times when the thread was waiting on Adabas I/O operations or other resources. The time, counted in 16-microsecond units, is called "Adabas thread time". The returned count is in binary format.

User Area

The user area field is reserved for use by the user program. When making logical user calls, the user area is neither written nor read by Adabas.

For compatibility with future Adabas releases, Software AG recommends that you set unused control block fields to null values corresponding to the field’s data type.

Programming Examples

Programming examples that show control block construction in each host language are provided in section *Programming Examples* of this documentation.

- *Examples for Assembler*
- *Examples for COBOL*
- *Examples for PL/ I*
- *Examples for FORTRAN*

Logging the Control Block

There are two formats for the command log. The newer format was first implemented in Adabas 5.2. The ADARUN parameter CLOGLAYOUT determines which format is used.

If CLOGLAYOUT=4, Adabas logs the following control block fields into the Adabas basic command logging area:

command code	command option 1
command ID	command option 2
file number	additions 2
response code	command time
ISN	

All other control block fields are logged into the extended command logging area.

If CLOGLAYOUT=5, all Adabas control block fields are logged in the basic command logging area. The extended logging area does not exist in this format.

The CLOGLAYOUT parameter is described in the *Adabas Operations* documentation. *Appendix A* in the *Adabas DBA Reference* documentation provides a detailed description of both command log formats.

Format and Record Buffers

The format buffer specifies the fields to be processed during the execution of an Adabas read/update command.

For read commands, the values of the fields specified in the format buffer are returned by Adabas in the record buffer.

Format Buffer	`AA,BB`	Name of the fields to be updated
Record Buffer	`value-AA value-BB`	Field values provided by user

For add/update commands, the new values for the fields specified in the format buffer are provided by the user in the record buffer.

Format Buffer	`XX,YY`	Name of the fields to be updated
Record Buffer	`value-XX value-YY`	Field values provided by user

For the OP command, the record buffer indicates the type of user and the files to be used.

The record buffer is also used for user data (OP, RE, CL, ET commands).

The format buffer must be long enough to hold the largest field definition contained in the related program, including the ending point (.).

Format Buffer Syntax

This section describes the syntax used to construct the format buffer.

Multiple record formats may be specified in the format buffer for files that contain various record formats. The specific record format to be used is determined by the value of a field in the file (for example, record type). When specifying multiple record formats, each one must be preceded by a format selection criterion.

The format buffer has the following syntax:

```
{ [ format-selection-criterion ] record-format } , ... .
```

A comma must be used to separate all format buffer entries. One or more spaces may be present between entries. The last entry may not be followed by a comma.

The format buffer must end with a period.

format-selection-criterion

The syntax of the format selection criterion is

```
( field-name operator { value , ... } )
```

field-name

- The field name used must be present in the FDT of the Adabas file being read.
- The field can be a multiple-value field.
- The field can be contained within a periodic group.
- NU or NC/NN option fields must have a non-null value; otherwise, the selection criterion is false.
- Group names, hyperdescriptors, and phonetic descriptors cannot be specified.

operator

EQ (or) =	equals
NE	not equal
LT (or) <	less than
GT (or) >	greater than
LE	less than or equal
GE	greater than or equal

value

The "value" is a numeric integer or an alphanumeric value. A series of values can be specified, separated by commas and using the EQ operator. An alphanumeric value must be enclosed within apostrophes (’value’).

Example:

```
(SA = 1) record-format-1, (SA = 2,3,4) record-format-2, (SA GE 5) record-format-3.
```

- If the value of field SA is 1, "record-format-1" is used;
- If the value of field SA is 2, 3 or 4, "record-format-2" is used;

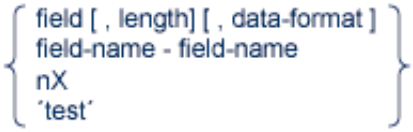
- If the value of field SA is equal to or greater than 5, "record-format-3" is used.

The first criterion that is met is used. If no criteria are met, a response code is returned.

record-format

The record-format is used to indicate which fields are to be read (read commands) or updated (update commands).

The syntax of the record format is



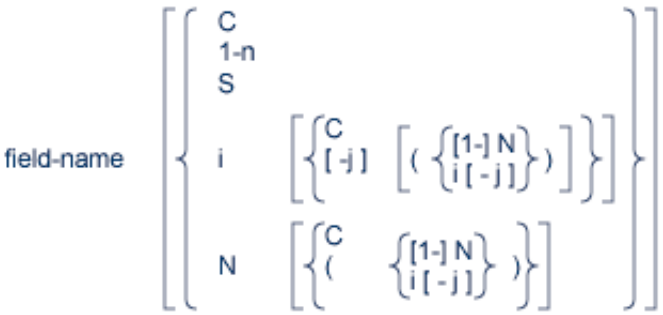
Record Format Syntax

The "field" term is explained in the following section. For information about the

- "length" and "data-format" terms, see page *Length and Data Format*.
- field series notation "field-name - field-name", see *Field Series Notation*.
- space notation "nX", see *Space Notation (nX)*.
- text insertion notation ’text’, see *Text Insertion Notation*.

field

The syntax for "field" is



Field Syntax

This section discusses the "field-name" term. See the following additional information:

- *Count Indicator (C)*.
- *Highest Occurrence/Value Indicator (N)*.
- *SQL Significance Indicator (S)*.

The "field-name" is the name of the field or group for which the value(s), range, or count is requested or for which a new value(s) is being provided. The name specified must be two characters in length and must be present in the FDT of the file being read/updated. The name can refer to an elementary, sub-/superfield, a multiple-value field, a group, or a periodic group.

A name that refers to a group results in all the fields within the group being referenced. Use of group names can greatly reduce the time required to process the command. A group name cannot be used if the group contains a multiple-value or variable-length field (no standard length).

For access commands, the same name may be specified more than once. In this case, the field value is returned multiple times.

For update commands, the same name cannot be used more than once (except in the case of multiple-value fields, as explained later in this section).

A sub-/superfield, or a sub-/superdescriptor name may be specified for access commands but not for update commands.

The following *Allowed Format Buffer Field Types* table illustrates the relationship between allowed single-value field type and type of command:

Cmd Type	Field	Subfield	Superfield	DE	SUBDE	SUPERDE	COLDE
Read (Ln) [1]	yes	yes	yes	yes	yes	yes	yes [3]
Add (Nn)	yes	no	no	yes	no	no	no
Find (Sn)	yes	yes	yes	yes	yes	yes	yes
Update (A1/4) [2]	yes	no	no	yes	no	no	no

Notes:

1. Format "C." may be used with read commands to request the record in its compressed format.
2. A field specified for an update command cannot
   be repeated;
   contain a "1-N"-type specification;
   specify a group and an element (field or group) contained in the group.
3. A collation descriptor (COLDE) can only be specified in the format buffer of the L9 command and only when the decode option has been specified in the user exit. The value returned is not the index value but the original field value.

For the Adabas file definitions used in all the examples in this section, see *File Definitions Used in Examples*.

Index or Range Notation

The following is the field name syntax for selecting multiple-value fields or occurrences of periodic groups:

```
field-name i [ - j ]    (periodic group or multiple-value field index or range)
```

where

i	is the periodic group or multiple-value occurrence
i-j	is the periodic group or multiple-value occurrence range

Periodic group names *must* be followed by a numeric or other appropriate suffix (see the discussions of the *Count Indicator (C)* and the highest occurrence/value indicator *Highest Occurrence/Value Indicator (N)* for more information). Specifying a periodic group name as simply "field-name" is incorrect syntax.

Multiple-value fields can be specified by explicitly identifying a particular value (indexing) or by referencing each value in sequence, letting Adabas assign an index based on the sequence. See *Multiple-Value Fields* for more information.

Example	Selects . . .
AA3	the third value of multiple-value field AA, the third occurrence of periodic group AA, or the first occurrence of the single- or multiple-value field AA contained within the third occurrence of a periodic group.
AA3-6	the third through sixth values of the multiple-value field AA, the third through sixth occurrences of periodic group AA, or the first occurrence of the (single or multiple-value) field AA contained within the third through sixth occurrences of a periodic group.

Periodic Groups

If a periodic group (or a field within a periodic group) is to be referenced, the specific occurrence to be used must be specified. This is accomplished by appending a one- to three-digit index (value 1-191, leading zeros are permitted) to the name.

An occurrence is identified by the name of the periodic group and the occurrence number.

Example	Selects . . .
GB3	the third occurrence of periodic group GB (fields BA3, BB3, BC3).

When selecting fields within one or more periodic groups, the field name and occurrences (values) are used; the group name is implied.

Example	Selects . . .
BB06	the sixth occurrence of field BB.

To refer to a range of occurrences of a periodic group (or a field within a periodic group), enter the periodic group or field name, followed by the first and last occurrence numbers separated by a hyphen. The occurrence numbers must occur in ascending sequence; a descending range is not permitted.

Example	Selects . . .
GB2-4	the second through fourth occurrences of periodic group GB, including all fields within these occurrences (BA2, BB2, BC2; BA3, BB3, BC3; and BA4, BB4, BC4).
BA2-4,BC2-4	the second through fourth occurrences of BA and BC (BA2, BA3, BA4, BC2, BC3, BC4).

Multiple-Value Fields

Multiple-value fields can be specified in the format buffer in two ways: by explicitly identifying a particular value (indexing) or by referencing each value in sequence, letting Adabas assign an index based on the sequence. These two methods apply as well to sub- or superdescriptors derived from multiple-value fields.

1. Indexing: To refer to a particular value of a multiple-value field, enter a one- to three-digit index (value 1-191, leading zeros permitted) after the field name.

Example	Selects . . .
MF2	the second value of the multiple-value field MF.
MF1, MF10	the first and tenth values of the multiple-value field MF.

   To refer to a range of values for a multiple-value field, specify the first and last values desired, connected by a hyphen, immediately after field name. An ascending range must be specified.

Example	Selects . . .
MF1-3	the first three values of the multiple-value field MF.

2. Sequencing: To refer to multiple-value fields by repeating the field name, first specify the first value, then the second, and so on.

Example	Selects . . .
MF,MF	the first and second values of the multiple-value field MF.
AA,MF,AB,MF,AC,MF	the first three values of the multiple-value field MF and the values for the fields AA, AB and AC, in the order shown.

Both methods of referencing a multiple-value field can be used in the same format buffer. If no occurrence index is specified, an index that is one higher than the last index, proceeding from left to right, is implicitly (or explicitly) assigned. If the last index was specified as "N" or "1-N", referring to the highest existing occurrence, a new field specification without an explicit index will refer to the *same* field occurrence as the last specification. See *Highest Occurrence/Value Indicator (N)* for more information about the highest occurrence/value indicator N/1-N/NC.

For an update command where *all* format buffer references to a multiple-value field are specified without indexes (i.e., by sequencing), only those occurrences specified will remain in the record; all other occurrences that might exist are deleted. If *any one* reference to a multiple-value field is specified with an index, then only the specified occurrences are changed; all other occurrences remain unchanged.

Multiple-Value Fields within Periodic Groups

If a multiple-value field is within a periodic group, specifying the multiple-value field name implies the periodic group name. The group name, therefore, does not have to be specified; only the group occurrence or occurrence range is required. The general syntax for selecting a multiple-value field value or value range within a periodic group occurrence or occurrence range is

```
field-name i    [ - j  ]  ( i [ -  j ] )
```

where

i	is the periodic group occurrence
i-j	is the periodic group occurrence range
(i)	is the multiple-value field value
(i-j)	is the multiple-value field value range

To refer to a multiple-value field contained within a periodic group, enter the occurrence number of the periodic group after the field name, followed by the desired multiple-value field values or range of values enclosed within parentheses.

Example	Selects . . .
CB2(5)	the fifth value of the multiple-value field CB in the second occurrence of the periodic group that contains field CB.
CB2(1-5)	the first five values of the multiple-value field CB in the second occurrence of the periodic group that contains field CB.

To refer to either the same multiple-value field or the same ranges of multiple-value fields contained within a range of periodic group occurrences, enter the range of occurrences after the field name, followed by the multiple-value field value or range of values, enclosed within parentheses.

Example	Selects . . .
CB3-5(2)	the second value of multiple-value field CB in each of the third through the fifth occurrences of the periodic group containing field CB.
CB1-2(1-4)	the first four values of the multiple-value field CB in the first occurrence of the periodic group containing field CB, followed by the first four values of CB in the second occurrence of the periodic group.

Count Indicator (C)

To obtain the count of periodic group occurrences, or the count of existing values of a multiple-value field not in a periodic group, specify the periodic group or multiple-value field name followed by "C":

```
field-name C     count for multiple-value field or periodic group "field-name"
```

Example	Selects . . .
GBC	the highest occurrence number (also the occurrence count) in periodic group GB.
MFC	the number of existing values in multiple-value field MF that are not contained in a periodic group.

The count is returned in the record buffer as a one-byte binary number unless an explicit length and/or format is specified (see *Length and Data Format*).

To obtain the count of the existing values of a multiple-value field contained within a periodic group, enter the multiple-value field name followed by the group occurrence and "C":

```
field-name i C   count for multiple-value field "field-name" within a group occurrence
```

Example	Selects . . .
CB4C	the number of existing values of multiple-value field CB in the fourth occurrence of a periodic group.

For update commands, specifying "C" causes Adabas to skip in the record buffer the number of positions occupied by the count field, thus ignoring the count.

The user cannot directly update multiple-value or periodic group count fields. These count fields are updated by Adabas when multiple-value field values and periodic group occurrences are added or deleted.

Highest Occurrence/Value Indicator (N)

The indicator "N" selects the last value in a series of values comprising a multiple-value field, or the last occurrence of a periodic group, removing the need to know the number of the last value or occurrence.

The notation "1-N" selects all values comprising a multiple-value field, or all occurrences of a periodic group. For multiple-value fields in periodic groups, it is not possible to combine the specification 1-N for the group occurrence with any specification for the field occurrences.

The notation "NC" selects the count of the existing values of a multiple-value field in the last occurrence of the periodic group containing the field.

`field-name N`	Last occurrence of periodic group "field-name" or the highest value of multiple-value field "field-name".
`field-name NC`	Count of values for multiple-value field "field-name" in the last occurrence of the periodic group containing "field-name".
`field-name NC`	Selects all values of multiple-value field "field-name" or all occurrences of the periodic group "field-name".
`field-name N (N)`	The last value of multiple-value field "field-name" in the last occurrence of the periodic group containing "field-name".
`field-name N(1-N)`	All values of multiple-value field "field-name" in the last occurrence of the periodic group containing "field-name".
`field-name N( i [-`	Value or range of values of multiple-value field "field-name" in the last occurrence of the periodic group containing "field-name".
`field-name i [- j ]`	The last value of multiple-value field "field-name" in an occurrence or range of occurrences of the periodic group containing "field-name".
`field-name i [-j ]`	All values of multiple-value field "field-name" in an occurrence or range of occurrences of the periodic group containing "field-name".

Example	**Selects . . .**
MFN	the highest (last) value of multiple-value field MF. If multiple-value field MF contains four values, the fourth value (the last value entered) is selected.
GBN	the highest occurrence number of a periodic group GB.
MFNC	the count of existing values for the multiple-value field MF in the highest occurrence of the periodic group containing MF.
MF1-N	all values of the multiple-value field MF.
GB1-N	all occurrences of periodic group GB.
MFN(1-N)	all values for the multiple-value field MF in the highest occurrence of the periodic group containing MF.
MFN(N)	the highest value for the multiple-value field MF in the highest occurrence of the periodic group containing MF.
MFN(4)	the fourth value of the multiple-value field MF in the highest occurrence of the periodic group containing MF.
CB3-5(N)	the highest value of the multiple-value field CB in the third through fifth occurrences of the periodic group containing CB.
CB2(1-N)	all values of the multiple-value field CB in the second occurrence of the periodic group containing CB.

SQL Significance Indicator (S)

The "S" significance, or null, indicator and the corresponding null indicator value in the record buffer indicate whether a field's value is significant, including zero or blank, or not significant (undefined). The S indicator can only be applied to elementary fields that are defined with the NC option, but not for an NU option field:

```
field-name S    SQL significance indicator
```

See the section *Record Buffer* for a description of the null value indicator, and the ADACMP utility description in the *Adabas Utilities* documentation for information about defining SQL null fields.

The related null indicator value in the record buffer, described below, has a two-byte standard length and fixed point format; this length and format cannot be overridden.

For update-type commands, a null value will be stored in the field if the corresponding null indicator value is X'FFFF' and no NN option is specified for the field. Otherwise, the null indicator must be X'0000'. This, combined with the S indicator, shows that the field's value given elsewhere in the record buffer is to be stored.

The S indicator is not required for update-type commands; if the S indicator is not specified, the field value is updated exactly as if the S indicator had been specified and the corresponding null indicator value had been set to zero (a null in the field is a value). See the examples below.

For read-type commands, the S indicator is required when the NC fields are defined without the NN option. If the S indicator is not present when a read command detects an NC-specified field and the field actually contains a null value, a response code 55 is returned.

Example:

This example uses the "field-name S" indicator with the two-byte null indicator in the record buffer to update or add a record, setting field AA equal to the SQL null value and ignoring the value for field AA:

Format Buffer	`AAS , AA , 2 , ... .`	Name of the fields to be read
Record Buffer	`FFFF  123F`	Field values returned by Adabas

For examples showing the use of the SQL significance indicator when a group or range of fields containing an NC field is specified, see the section *The SQL Significance Indicator and Field Series Notation*.

The "field-name S" indicator can be anywhere within the format buffer; that is, it need not precede the corresponding field element. For update-type commands, the format buffer cannot contain more than one element referring to the same field:

AAS.	valid for read/update commands
AA.	valid for read/update commands
AAS,AA,AAS.	valid for read commands only

This means that several format buffer elements referring to the same field cannot be specified for an update-type command:

AA,AA.	causes response code 44
AA,AAS,AA.	causes response code 44

Field Series Notation

The notation "field-name - field-name" may be used to refer to a series of consecutive fields (as ordered in an FDT). The user specifies the beginning and ending field names connected by a dash:

```
field-name - field-name  series notation
```

No multiple-value field or periodic group may be contained within the series.

A name that refers to a group may not be specified as the beginning or ending name, but a group may be embedded within the series.

Standard format and length is in effect for all the fields within the series. No length or format override is permitted.

Example	Selects . . .
AA-AC	the fields AA, AB, and AC.
AA-GC	nothing. The series may not contain a multiple-value field or a periodic group.
GA-AC	nothing. The series may not begin/end with a group.
AA,5,U,-AD	nothing. A length and/or format override is not permitted in a series notation.

The SQL Significance Indicator and Field Series Notation

When a group or range of fields contains a field specified with the NC option, the corresponding S operator is optional for read (Lx) commands. For update (A1) commands, the S operator must not be specified. Adabas assumes that the null indicator corresponding to the NC field in the format buffer is located just in front of the field’s value in the record buffer.

For example, given the following field definitions in the FDT:

```
01,GR
  02,AA,8,A
  02,BB,8,A,NC
  02,CC,8,A
```

if the format buffer of an update-type command specifies GR. or AA-CC. , the record buffer has the following structure:

```
AA-value null-indicator-BB BB-value CC-value
```

That is, the null indicator must be included in the record buffer sequence, although the S indicator was not (and must not be) specified in the format buffer.

If the format buffer of a read (Lx) command specifies GR,BBS. or AA-CC,BBS., the record buffer has the following structure:

```
AA-value null-indicator-BB BB-value CC-value
null-indicator-BB
```

In other words, the first appearance of the null indicator is implied in the record buffer while the second appearance was explicitly called for by the format buffer.

Length and Data Format

The length and format parameters are used if a field value is being provided or is to be returned in a length and/or format different from the standard defined for the field in the FDT. If the length and/or format parameters are omitted, the field value must be provided or is returned in the standard length and format of the field:

```
[ , length ] [ , data-format ]
```

Possible format/length conversions are suggested by the information in the following *Permitted Data Lengths and Formats* table. A format conversion cannot be specified for subfields or subdescriptors; superfields or superdescriptors; or hyperdescriptors.

Fmt	Max Length (in bytes)	Data Type	Compatible Formats
A	253	alphanumeric, left-justified	W
W	253 [1]	wide-character, left-justified	A
A,W	16,381 [1,2]	alphanumeric or wide-character with LA (long alpha) option; left-justified; preceded by optional two-byte binary (inclusive) length	W,A
B	126	binary; right-justified; unsigned	A,F,P,U
F	4	fixed-point; right-justified; signed; two or four bytes	A,B,P,U
G	8	floating-point; four or eight bytes	none
P	15	packed decimal; signed; positive=A,C,E, or F; negative=B or D	A,B,F,U
U	29	unpacked decimal; signed; positive=A,C,E, or F; negative=B or D	A,B,F,P

Notes:

1. Like an alphanumeric field, a wide-character field may be a standard length in bytes defined in the FDT, or variable length. Any non-variable format override for a wide-character field must be compatible with the user encoding; for example, a user encoding in Unicode requires an even length (max. 252 bytes).

2. Maximum long alpha length if the length (variable field length notation) precedes the field in the record buffer; otherwise, the maximum length is 253 bytes.

The length specified must be large enough to contain the value in the chosen format, but cannot exceed the maximum length permitted.

If a length of zero is specified, or if "field-name" refers to a variable-length field (no standard length), the value returned by Adabas in the record buffer is preceded by a one-byte binary field containing the length of the value (including the length byte itself). For update commands, you must provide this length byte at the beginning of the record buffer.

The format specified must be compatible with the standard format of the field.

- Conversion between packed/unpacked decimal values and binary is limited to values between 0 and 2,147,483,647.
- Conversion from a numeric format to alphanumeric results in an unpacked value, left justified, without leading zeros and with trailing blanks. For example, the three-byte packed value "10043F" would be converted to "F1F0F0F4F3404040". Value truncation is possible with this type of conversion.

Edit Mask Notation (Read Operations Only)

Edit masks are used according to the standard edit mask rules used in the COBOL programming language.

An edit mask may only be specified for numeric fields. All data returned by Adabas to an edited field is converted to unpacked decimal format regardless of the standard format of the field. A maximum of 15 digits (not counting edit characters) can be returned to an edited field.

For a field with an edit format specified, the length parameter must be large enough to contain the field value plus all required edit characters.

Format	Generates the edit mask . . .
E1	zzzzzzzzzzzzzzz
E2	zzzzzzzzzzzzz9-
E3	zzzzzzzzz99.99.99
E4	zzzzzzzzz99/99/99
E5	z.zzz.zzz.zzz.zzz,zz
E6	z,zzz,zzz,zzz,zzz.zz
E7	z,zzz,zzz,zzz,zz9.99-
E8	z.zzz.zzz.zzz.zz9,99-
E9	*,***,***,***,**9.99-
E10	*.***.***.***.**9,99-
E11	user-designated mask
E12	user-designated mask
E13	user-designated mask
E14	user-designated mask
E15	user-designated mask

Note:
Although edit formats E3 and E4 provide space for the century digits (see the following examples), they do not *enforce* date formats that are compatible with year 2000 requirements.

Examples:

Format Buffer	Field Value	Edited Value
XC,15,E1.	009877	bbbbbbbbbbb9877
XC,8,E4.	301177	30/11/77
XB,5,E7.	-366	3.66-
XB,7,E9.	542	**5.42
Y2,10,E4.	20000229	2000/02/29

Space Notation (nX)

The nX syntax is used differently for read and update commands:

nX

For read commands, nX indicates that "n" spaces are to be inserted in the record buffer by Adabas immediately before the next field value:

Format Buffer	**AA , 5X , BB**	Name of the fields to be read
Record Buffer	value-AA 5-blanks value-BB	Field values returned by Adabas

For update commands, nX causes "n" positions in the record buffer to be ignored by Adabas:

Format Buffer	**XX , 5X , YY**	Name of the fields to be updated
Record Buffer	value-XX ignore-5-bytes value	Field values provided by user

Text Insertion Notation

The 'text' syntax is used differently for read and update commands:

'text'

For read commands, the character string specified in the format buffer is to be inserted in the record buffer immediately before the next field value. The character string provided can be 1-255 bytes long, and may contain any alphanumeric character except a quotation mark.

For example:

Format Buffer	**AA , 'text' , BB**	Name of the fields to be read
Record Buffer	value-AA text value-B	Field values returned by Adabas

For update commands, the number of positions enclosed within the apostrophes in the format buffer will be ignored in the corresponding positions of the record buffer.

Format Buffer Performance Considerations

Performance improvements may be achieved by using the following guidelines during format buffer construction:

- Use group names wherever possible rather than referring to elementary fields individually. Use of group names reduces the time required by Adabas to interpret the format buffer.
- Use of the field series notation does not result in performance improvements. A field series notation is converted by Adabas into a series of elementary fields.
- Use length and format overrides only when necessary. Using overrides requires additional processing time when interpreting the format buffer and when processing the field.
- If the same fields of a record are to be read and then updated, the same format buffer should be used for the read and update commands. For more information, refer to the descriptions of command and format IDs beginning on page .

- You should request periodic (PE) group and multiple-value (MU) field occurrences, based on the requirements of the application. In other words, do not arbitrarily request all occurrences; doing so requires extra time to translate the format buffer, and may mean decompressing numerous empty occurrences. Generally, the AA1-N field argument is the most efficient for selecting periodic group occurrences.

Record Buffer

The record buffer is used primarily with read, search, and update commands.

For read commands, Adabas returns the requested field values in this buffer in the order specified by the format buffer. Each value is returned in the standard length and format defined for the field unless a length and/or format override was specified in the format buffer. If the value is a null value, it is returned in the format that is in effect for the field, as follows:

Field Type	**Null value represented by . . .**
Alphanumeric (A)	blanks (hex ’40’) or blank of user override encoding
Binary (B)	binary zeros (hex ’00’)
Fixed (F)	binary zeros (hex ’00’)
Floating Point (G)	binary zeros (hex ’00’)
Packed (P)	decimal packed zeros with sign (hex ’00’ followed by ’0A’, ’0B’, ’0C’, ’0D’ or ’0F’ in the rightmost, low-order byte)
Unpacked (U)	decimal unpacked zeros with sign (hex ’F0’ followed by ’C0’ or ’D0’ in the rightmost, low-order byte)
Wide-character (W)	Unicode blanks (hex ’20’) or blank of user override encoding

Note:
SQL-compatible null values in NC/NN option fields require the additional null value and significance indicator. See *Specifying and Reading the SQL Null Indicator*, and *SQL Significance Indicator (S)*.

Adabas returns the number of bytes equal to the combined lengths (standard or overridden) of all requested fields.

When updating a record, you must specify the new value in the record buffer. If a null value is being provided, it must be provided according to the field type in effect, as described above.

The record buffer is also used to transfer information between the user program and Adabas in the following commands:

Command	Data Provided	Data Returned
OP	Files to update and the operation type (ET, exclusive control)	User data (optional)
LF	-	Field definitions for the file
RE	-	User data stored in system file
C5	Protection log user data	-
ET/CL	User data (optional)	-

Specifying and Reading the SQL Null Indicator

To support Software AG's Adabas SQL Server (ESQ) and other structured query languages (SQLs), fields defined with the NC/NN (not-counted/null-not-allowed) options indicate an SQL-significant null with a two-byte binary null indicator in the record buffer.

Whether a field's "zero" value is significant or an irrelevant null (unspecified) depends on the null indicator specified in the record buffer when the value is entered or changed, or returned in the record buffer when the value is read.

In addition to specifying or reading the value itself, either

- set the null indicator into the record buffer position that corresponds to the field's designation in the format buffer for an update operation, or
- ensure that your program examines the null indicator (if any) returned in the record buffer position corresponding to the field's position in the format buffer for a read operation.

The null indicator is always two bytes long and has fixed-point format, regardless of the data format.

For a read (Lx) or find with read (Sx with format buffer entry) command, the null indicator value returns one of the following (hexadecimal) null indicator values, according to the actual value that the selected field contains

FFFF	a null value in this field is not significant.
0000	a null value in this field is a significant value; that is, a true zero or blank.
xxxx	the field was truncated. The null indicator contains the length (xxxx) of the entire value as stored in the database record.

For an update (Ax) or add (Nx) command, the (hexadecimal) null indicator value in the record buffer must be set to

FFFF	the field value is set to "undefined", an insignificant null; the field's contents in the record buffer is irrelevant when set to binary zero or blank characters.
0000	if either no value is specified in the record buffer, or binary zero or blanks are specified, the field contains a significant null value.

For an *add* command, if no value for the field is supplied in the record buffer for a field defined with the NC option, the field is treated as a null field. The following example shows how a null would be represented in a two-byte Adabas binary field AA defined with the NC option:

Field definition: 01,AA,2,B,NC

	For a nonzero value	**For a blank**	**For null**
Null Value indicator in Record Buffer	0 (binary value is significant)	0 (binary null is significant)	FFFF (binary null is not significant)
Data	0005	0000 (zero)	not relevant
Adabas internal representation	0205	0200	C1

For an *update* (A1/N1) command, the field value is always significant whenever the field is defined with the NC option; the field is treated as if a hexadecimal null indicator value of "0000" has been specified.

For a *read* command, if the null indicator is not specified for an NC option field, the field value is returned in the record buffer whenever there is a significant value in the record. If the Data Storage record contains a "not significant" (FFFF) indicator value for the field, response code 55 will be returned when the record is read.

Specifying a Field with LA (Long Alpha) Option

The LA option is normally used with variable-length data. The following specifications illustrate alpha fields with LA option in the format and record buffers.

The length of an alpha field with LA option can be specified in the record buffer. The field value is preceded by a two-byte length field containing the length of the value, plus 2 (inclusive length).

Format Buffer	`AA, ...`	Name of the fields to be read

Record Buffer	`0005ABC ...`
	or
	`2712 ...(10,000 characters)..`
		Field values returned by Adabas

The length of an alpha field with LA option can also be specified in the format buffer; however, the length is then limited to 253 bytes:

Format Buffer	`AA,5, ...`	Name of the fields to be read
Record Buffer	`ABC__`	Field values returned by Adabas

A field with LA option can also have the NU (null suppression), NC/NN SQL null significance (page), or the MU (multiple-value field) option, and can be a member of a PE (periodic) group.

A field with the LA option cannot be a descriptor and cannot be the parent of sub-/superfields, sub-/superdescriptors, hyperdescriptors, or phonetic descriptors.

A field is compressed the same way, with or without the LA option; that is, by removing trailing blanks. This must be kept in mind if you store binary data in a long alpha field.

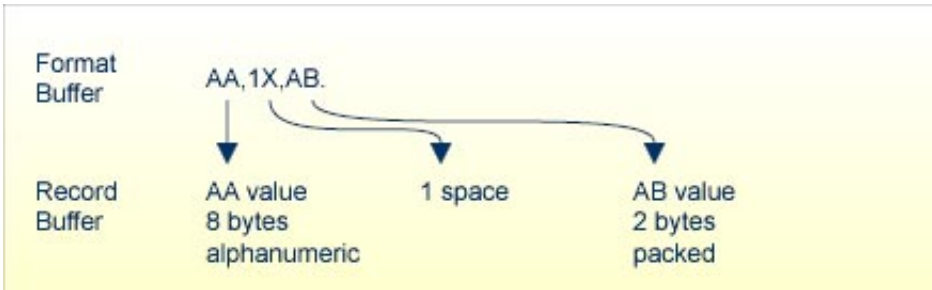
Format and Record Buffer Examples

This section provides examples of format and record buffer construction. For the Adabas file definitions used in all the examples in this section, see *File Definitions Used in Examples*.

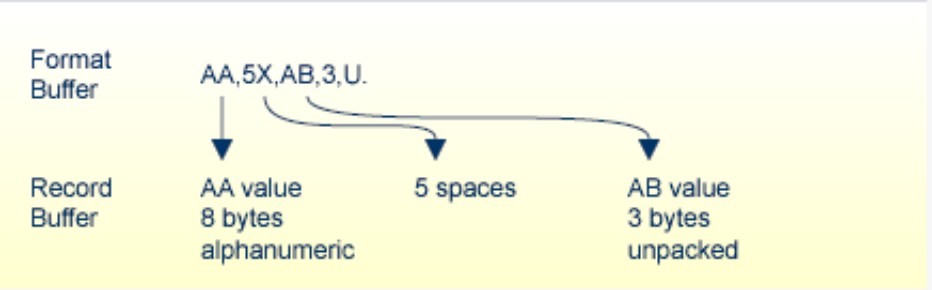
This section covers the following topics:

- Example 1: Using Elementary Fields (Standard Length and Format)
- Example 2: Using Elementary Fields (Length and Format Override)
- Example 3: A Reference to a Periodic Group
- Example 4: The First Two Occurrences of Periodic Group GB
- Example 5: The Sixth Value of the Multiple-Value Field MF
- Example 6: The First Two Values of the Multiple-Value Field MF
- Example 7: The Highest Occurrence Number of a Periodic Group GC and the Existing Number of Values for the Multiple-Value Field MF

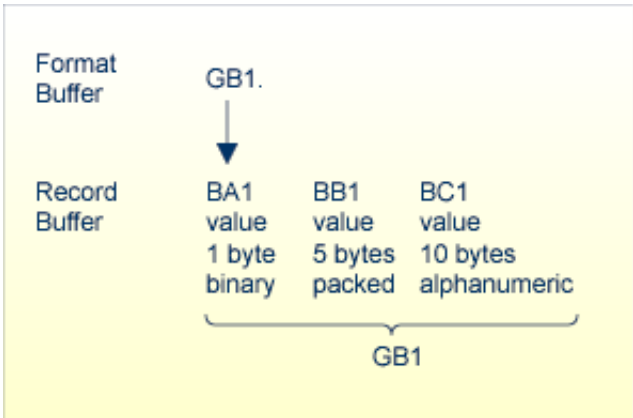
Example 1: Using Elementary Fields (Standard Length and Format)



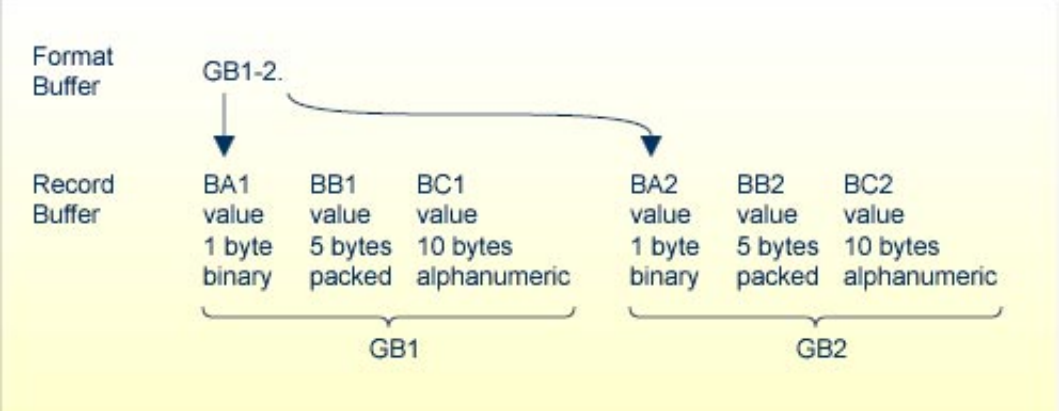
Example 2: Using Elementary Fields (Length and Format Override)



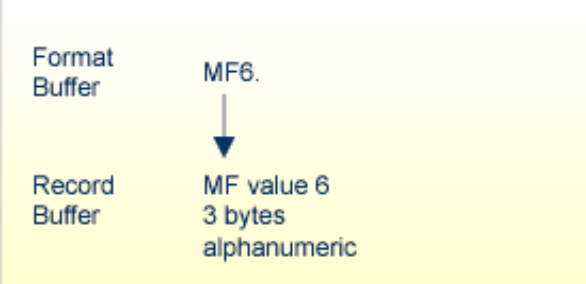
Example 3: A Reference to a Periodic Group



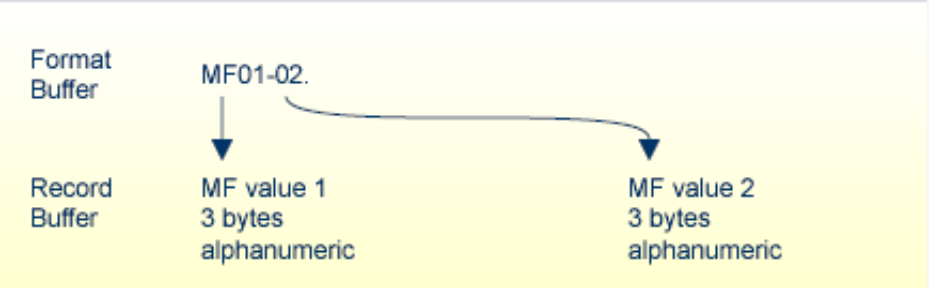
Example 4: The First Two Occurrences of Periodic Group GB



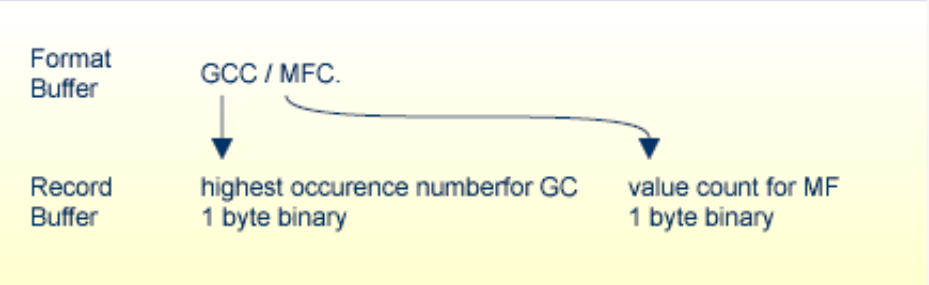
Example 5: The Sixth Value of the Multiple-Value Field MF



Example 6: The First Two Values of the Multiple-Value Field MF



Example 7: The Highest Occurrence Number of a Periodic Group GC and the Existing Number of Values for the Multiple-Value Field MF



Search and Value Buffers

The search and value buffers are used together to define

- the search criterion to select a set of records using a FIND command (S1, S2, S4); and
- the range of values to be traversed by logical sequential read commands (L3/6, L9).

The user provides the search expression(s) in the search buffer and the values which correspond to the search expressions in the value buffer.

The syntax used for the search buffer depends on the type of search criteria to be employed:

1. Single file search. The search criteria consists of one or more fields contained in a single file;
2. Multiple file search using physically coupled files. The search criteria consists of fields contained in two or more files which have been physically coupled using the ADAINV utility;

3. Search using the soft coupling feature. This feature provides for a combination of search, read, and internal list matching.

A search criteria may also contain one or more fields which are not defined as descriptors. If nondescriptors are used, Adabas performs a read operation to determine which records are to be returned to the user.

Note:
On files containing a large number of records, performing a search using nondescriptors can result in excessive response times.

Search Buffer Syntax

Overview

This section outlines the syntax statement options for the search buffer. Delimiters (commas, slashes, parentheses, semicolons) must separate all search buffer entries as indicated. One or more spaces may be present between entries. The search statement must end with a period.

This section covers the following topics:

- Search One File
- Search Multiple, Physically-Coupled Files
- Search One or More Files Using Soft Coupling

Search One File

The following syntax statement is relevant when searching fields in a single file:

```
search-expression [ { , connecting-operator , search-expression } ...] .
```

For the syntax of the "search-expression", see *Search Expression*.

For information about the "connecting-operator", see *Connecting Search Expressions*.

Search Multiple, Physically-Coupled Files

The following syntax statement is relevant for multiple-file searches in which fields from two or more physically coupled files are to be used:

```
/ file-x / search-expression [ {, connecting-operator , search-expression } ... ]
{ , D , / file-y  search-expression /[ { , connecting-operator , search-expression }... ] } ... .
```

For information about search buffer syntax using physically-coupled files, see *Physically Coupled Files*.

Search One or More Files Using Soft Coupling

The following syntax statement is relevant for searching one or more files using soft coupling:

```
( m-file, m-field, s-file, s-field  [ { ; m-file, m-field, s-file, s-field } ... ])
/ s-file-x / search-expression [ { , connecting-operator , search-expression } ... ]
[ { , D ,  / s-file-y /search-expression [ { , connecting-operator , search-expression } ... ] } ... ] .
```

For information about search buffer syntax using soft coupling, see *Soft Coupling*.

Search Expression

The search expression syntax is common to all types of searches:

Search Expression Syntax

The search expression comprises either a field name with optional entries or a command ID.

A command ID value (enclosed within parentheses) identifies a list of ISNs resulting from a previous Sx command that specified the save-ISN-list option.

This section covers the following topics:

- field-name
- Occurrence Index [i]
- S (Significance) and Null Indicators
- length and format
- value-operator

field-name

The search expression can name a field (descriptor or nondescriptor), subdescriptor, superdescriptor, hyperdescriptor, collation descriptor, or phonetic descriptor. When using nondescriptors, multiple-value fields are permitted, but sub-/superfields are not.

If a nondescriptor is used, Adabas reads the entire file in order to determine which records satisfy the search criteria. If only descriptors are used, the inverted lists are used and no reading of records is necessary. Search criteria containing nondescriptors and descriptors may be combined.

If a descriptor field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons and therefore, the record will not be returned in a search. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

If the descriptor is defined with the NU option (null-value suppression), null values are not stored in the inverted lists; therefore, a search for all the records which have the null value will always result in no records found (even if there are records in Data Storage which contain a null value for the descriptor). This rule also applies to subdescriptors. A superdescriptor value is not stored if any field from which it is derived is defined with the NU option *and* the value of that field is actually null.

Occurrence Index [i]

The occurrence index (i) identifies a particular occurrence of a descriptor or nondescriptor within a periodic group and is used to limit the search to only the values located in the specified occurrence. If no index is provided, the values in all occurrences are searched.

- The index comprises one to three digits; leading zeros are permitted.
- An index is *not* permitted for
  - a superdescriptor derived from a field within a periodic group; or
  - a descriptor that is a multiple-value field, or a sub-/superdescriptor derived from a multiple-value field.

In these cases, the values in all occurrences of the periodic group are searched.

S (Significance) and Null Indicators

For fields with the SQL null value compression option NC, a selection for "null" or "not null" can also be made using an "S" null indicator element similar to that described in the section *The SQL Significance Indicator and Field Series Notation*.

Note:
The NC option cannot be applied to fields with the NU (null-value suppression), FI (fixed storage), MU (multiple-value), or PE (periodic group) options, or to group fields.

The SQL significance ("S") indicator must be added to the field name ("field-nameS") and the corresponding SQL null indicator must be specified in the value buffer.

The following hexadecimal null indicators are allowed as search argument values:

FFFF	select null values
0000	select non-null values

Any other null indicator value causes an Adabas response code 52.

The null indicator (hexadecimal FFFF or 0000) has a standard length of two bytes and fixed-point format; this length and format cannot be overridden.

The "S" indicator can only be used with the equals (=) value-operator; using S with any other value operators causes an Adabas response code 61.

Examples:

The S significance operator is part of the search argument for the field AA.

```
AAS.
```

Select records with the FN field value of packed +1 and the AA field value of null (undefined):

Search Buffer	`FN , 2 , P , D , AAS.`	Search argument
Value Buffer	`001F FFFF`	Field value specification

Note:
Insignificant null values are not stored in the index. This can cause a search-for-null operation to be quite costly for an application program's performance.

Select records with the FN field value of packed +1 and the AA field value of non-null:

Search Buffer	`FN , 2 , P , D , AAS.`	Search argument
Value Buffer	`001F 0000`	Field value specification

length and format

The "length" and "format" of the field/descriptor value as provided in the value buffer may be stated explicitly with these parameters. If the length or format parameter is omitted, the value in the value buffer must comply with the standard length and format of the field/descriptor, as shown in the following *Permitted Data Lengths and Formats* table.

value-operator

A value-operator indicates the logical operation to be performed between the preceding descriptor and its corresponding value in the value buffer.

The following operators may be specified:

EQ (or) =	equals
GE	greater than or equal to
GT (or) >	greater than
LE	less than or equal to
LT (or) <	less than
NE	not equal to

If no value-operator is specified, an "equals" operation is assumed.

Examples:

The following examples show the use of a value-operator:

AA.	AA equals the value specified in the value buffer (the default)
AA,LT.	AA is less than the value specified in the value buffer
AA,GE.	AA is greater than or equal to the value specified in the value buffer.

The NE (not equal to) operator selects all records with the FN field not equal to "MIKE":

Search Buffer	`FN,4,A,NE.`	Search argument
Value Buffer	`MIKE`	Field value specification

Replacing the NE operator in the above example with EQ (equal to) would select only FN field values of "MIKE".

Connecting Search Expressions

A "connecting-operator" may be used to connect search expressions. The permissible connecting operators are as follows:

Operator	Description
D	The results of two search expressions are to be combined using a logical AND operation: `AA,D,AB.`
O	The results of two search expressions are to be combined using a logical OR operation. The OR operator may only be used to connect search expressions which use the same descriptor: `AA,O,AA.` **valid** `AA,O,AB.` **invalid**
R	Fields or command IDs that point to ISN lists derived from different descriptors are to be combined using a logical OR operation: `AA,5,A,R,AB,LT.` **valid**
S	A FROM-TO range (inclusive) which involves two search expressions. The same descriptor must be used in both expressions: `AA,S,AA.` **valid** `AA,S,AB.` **invalid**
N	Excludes a single value or a range of values from the immediately preceding FROM-TO range. This operator can only be specified in conjunction with the "S" operator, and must apply to the same field specified in the FROM-TO range. Phonetic descriptors cannot be specified: `AA,S,AA,N,AA.` **valid** `AA,S,AA,N,AB.` **invalid** `AA,S,AA,N,AA,S,AA.` **valid**
Y	The results of any number of D, O, R, S, and N search operations can be combined using a logical AND operation: `AA,D,AB,Y,AA,O,AA,Y,AA,S,AA,N,AA,S,AA.` The Y connecting operator functions like parentheses: only one level is allowed; that is, nested parentheses are not supported. All search expressions connected with the Y operator must apply to the same file.

See the section *Search/Value Buffer Examples* for more examples.

Processing Order for Search Buffer Connecting Operators

If different operators are used within a single search buffer argument, the operators are processed in the following order:

1. Evaluate all "S/N/O" operations, as described in this documentation;
2. Evaluate all "D" operations, if needed;
3. Evaluate all "R" operations, if needed;
4. Evaluate all "Y" operations, if needed.

Example:

```
Search Buffer = AA,S,AA,O,AA,D,AB,R,AC,D,AD.
```

is equivalent to

```
( ( (AA,S,AA),O,AA),D,AB),R,(AC,D,AD)
```

```
Search Buffer =  AA,D,AB,Y,AA,O,AA,Y,AA,S,AA,N,AA,S,AA.
```

is equivalent to

```
(AA,D,AB),Y,(AA,O,AA),Y,((AA,S,AA),N,(AA,S,AA))
```

Physically Coupled Files

The search buffer for a multiple-file search in which fields from two or more physically coupled files are to be used is

```
/ file-x / search-expression [ {, connecting-operator , search-expression } ... ]
{ , D , / file-y / search-expression [ { , connecting-operator , search-expression } ... ]} ... .
```

The only connecting operator allowed in search expressions for physically coupled files is the AND (D) symbol.

The search expressions can be in any order. The ISN values actually returned are from the coupled file specified by the Adabas control block’s file number field; this file is called the "primary" file.

The file number can appear only once for a given file. The file number must immediately precede the search expression (or expressions) referring to that file. A maximum of five (5) files may be specified in a single find command.

All files specified must have been previously coupled using the COUPLE function of the ADAINV utility.

All other syntax elements are entered as described in the sections *Search Expression* and *Connecting Search Expressions*.

Example:

Find the ISNs of all the records in file 1 that contain the three-byte (length override) unpacked decimal (format override) value "+20" in their AB fields, and that are coupled to records in file 2 containing the value "ABCDE" for field RB, which has a standard length of ten bytes and an alphanumeric format.

Search Buffer	`/1/AB,3,U,D,/2/RB.`	Search argument

Value Buffer	character-notation
	`020ABCDEbbbbb`
	hex-notation
	`F0F2F0C1C2C3C4C54040404040`
		Field value specification

Soft Coupling

The search buffer for a search in which soft coupling is to be used is constructed as follows:

```
(m-file, m-field, s-file, s-field  [ { ; m-file, m-field, s-file, s-field } ... ] )
/ s-file-x / search-expression [ { , connecting-operator , search-expression } ... ]
[ { , D ,  / s-file-y / search-expression [ { , connecting-operator , search-expression } ... ] } ... ] .
```

m-file, m-field

"m-file" is the main file. This file must also be specified in the file number field of the Adabas control block. The final resulting ISN list will include ISNs contained in the main file only.

"m-field" is the field in the main file that is to be used as the soft-coupling link field. This field must be a descriptor, subdescriptor, superdescriptor, or hyperdescriptor. It may not be a long alphanumeric field, or be contained within a periodic group.

s-file, s-field

"s-file" is the search file; "s-field" is a field within the search file. For each ISN selected from this search file (according to the search criterion), the field specified as "s-field" will be read. The value of the field will then be used to determine which ISNs in the main file have a matching value.

The field may be a descriptor or nondescriptor; it can be a subdescriptor, superdescriptor, hyperdescriptor, or a long alphanumeric field. It must have the same format as the corresponding "m-field". The standard length may be different. The field may not be contained within a periodic group.

A maximum of 42 soft-coupling criteria may be specified. Search criteria for soft coupling are illustrated in the section *Search/Value Buffer Examples*, examples 15-18.

Value Buffer

In the value buffer, the user specifies the values for each descriptor specified in the search buffer.

If the search expression is a command ID, no corresponding entry is made in the value buffer.

The values provided must be in the same sequence as the corresponding search expressions specified in the search buffer. All values provided must correspond to the standard length and format of the corresponding descriptor unless the user has explicitly overridden the standard length or format in the search buffer.

No intervening blanks or other characters such as a comma can be inserted between values in the value buffer. A period is not required to end the value buffer entry.

SQL Null Values and Indicators

When searching for fields defined with the NC (SQL null "not counted") option, the search buffer field definition must contain a null significance ("S") indicator and the corresponding value buffer argument value must display a two-byte binary null value indicator. See the section *S (Significance) and Null Indicators* for more information and examples of the null value indicator in the value buffer.

Sign Handling

Binary values are treated as unsigned numbers. Fixed-point, unpacked, and packed values are treated as signed numbers. Valid signs which may be provided are as follows:

Fixed

The sign is contained in bit 0 (high-order bit):

```
0 = positive
1 = negative (two’s complement)
```

Example (hex notation with decimal equivalent):

```
00000005 = +5
FFFFFFFB = -5
```

Unpacked

The sign is contained in the four high-order bits of the low-order byte:

```
C or A or F or E = positive (CAFE)
B or D           = negative (BD)
```

Example (hex notation with decimal equivalent):

```
F1F2F3  = +123
F1F2D3  = -123
```

Packed

The sign is contained in the four low-order bits of the low-order byte:

```
A or C or E or F = positive*
B or D           = negative*
```

* If a search value is being provided for a superdescriptor which is derived from a packed field, an F positive sign or a D negative sign must be provided.

Example (hex notation with decimal equivalent):

```
X’123F’ = +123
X’123C’ = +123
X’123D’ = -123
```

Search/Value Buffer Examples

This section contains examples of search and value buffer construction. For the Adabas file definitions used in all the examples in this section, see *File Definitions Used in Examples*. The values for the value buffer are shown in character and/or hexadecimal notation.

Example 1: Using a Single Search Expression

A search that uses a single search expression.

Select the ISNs of all the records in file 1 that contain the value "12345" for field AA, which has a standard length of eight bytes and numeric format.

Search Buffer	`AA.`	Search argument

Value Buffer	character-notation
	`00012345`
	hex-notation
	`F0F0F0F1F2F3F4F5`
		Field value specification

The same search may be performed using "AA,5." (length override) in the search buffer and the value "12345" (without trailing blanks) in the value buffer.

Example 2: Using Search Expressions Connected by AND

A search that uses two search expressions connected by the AND operator.

Select the ISNs of all the records in file 1 that contain the value "12345678" for the field AA, which has a standard length of eight bytes and numeric format, and the value "+2" for the field AB, which has a standard length of two bytes and packed decimal format.

Search Buffer	`AA,D,AB.`	Search argument
Value Buffer	hex-notation `F1F2F3F4F5F6F7F8002C`	Field value specification

Select the ISNs of all the records in file 1 that contain the value "12345678" for the field AA, which has a standard length of eight bytes and numeric format, and the value "+2" for the field AB, which has a length of three bytes (override) and unpacked decimal (override) format.

Search Buffer	`AA,D,AB,3,U.`	Search argument

Value Buffer	character-notation
	`12345678002`
	hex-notation
	`F1F2F3F4F5F6F7F8F0F0F2`
		Field value specification

This second search produces the same result as the first search, but shows the use of the length and format override in the search buffer.

Example 3: Using Search Expressions Connected by OR

A search that uses three search expressions connected by the OR operator.

Select the ISNs of all the records in file 2 that contain any of the values "284", "285", or "290" for the field XB. Length and format overrides are used.

Search Buffer	`XB,3,U,O,XB,3,U,O,XB,3,U.`	Search argument

Value Buffer	character-notation
	`284285290`
	hex-notation
	`F2F8F4F2F8F5F2F9F0`
		Field value specification

Example 4: Using Search Expressions Connected by FROM-TO

A search that uses two search expressions connected by the FROM-TO operator.

Select the ISNs of all the records in file 2 that contain any value in the range "+20" through "+30" for the field XB.

Search Buffer	`XB,S,XB.`	Search argument
Value Buffer	hex-notation `020C030C`	Field value specification

Example 5: Using Search Expression with BUT-NOT

A search that uses three search expressions connected by the FROM-TO and BUT-NOT operators.

Select the ISNs of all the records in file 2 that contain any of the values in the range "+20" through "+30" but not "+27" for the field XB.

Search Buffer	`XB,S,XB,N,XB.`	Search argument
Value Buffer	hex-notation `020C030C027C`	Field value specification

Select the ISNs of all the records in file 2 that contain any of the values in the range "+1" through "+200" or "+500" through "+600" for the field XB.

Search Buffer	`XB,S,XB,O,XB,S,XB.`	Search argument
Value Buffer	hex-notation `001C200C500C600C`	Field value specification

Example 6: Using a Multiple-Value Descriptor

A search in which a multiple-value field is used.

Select the ISNs of all the records in file 1 that contain the value "ABC" for any value of the multiple-value field MF.

Search Buffer	`MF.`	Search argument

Value Buffer	character-notation
	`ABC`
	hex-notation
	`C1C2C3`
		Field value specification

It is not possible to limit the search to a specific occurrence of a multiple-value field. The following search buffer entry is invalid:

Search Buffer	`MF2.`	Search argument

Example 7: Using a Descriptor Within a Periodic Group

A search in which a descriptor within a periodic group is used.

Select the ISNs of all the records in file 1 that contain the value "4" in any occurrence of the descriptor BA (which is contained in a periodic group).

Search Buffer	`BA.`	Search argument
Value Buffer	hex-notation `04`	Field value specification

Select the ISNs of all records in file 1 that contain the value "4" in the third occurrence of the descriptor BA (which is contained within a periodic group).

Search Buffer	`BA3.`	Search argument
Value Buffer	hex-notation `04`	Field value specification

Example 8: Using a Subdescriptor

A search that uses a subdescriptor. SA is a subdescriptor derived from the first four bytes of the field RA.

Select the ISNs of all the records in file 2 that contain the value "ABCD" for the subdescriptor SA.

Search Buffer	`SA.`	Search argument
Value Buffer	hex-notation `C1C2C3C4`	Field value specification

Example 9: Using a Superdescriptor with Alphanumeric Format

A search that uses a superdescriptor with alphanumeric format. SB is a superdescriptor derived from the first eight bytes of the field RA and the first four bytes of the field RB.

Select the ISNs of all the records in file 2 that contain the value "ABCDEFGH1234" for the superdescriptor SB.

Search Buffer	`SB.`	Search argument
Value Buffer	hex-notation `C1C2C3C4C5C6C7C8F1F2F3F4`	Field value specification

Example 10: Using a Superdescriptor with Binary Format

A search that uses a superdescriptor with binary format. SC is a superdescriptor derived from the fields XB and XC.

Select the ISNs of all the records in file 2 that contain the value "+20" for the field XB and the value "123456" for the field XC.

Search Buffer	`SC.`	Search argument
Value Buffer	hex-notation `020FF1F2F3F4F5F6`	Field value specification

Example 11: Using Previously Created ISN Lists

A search that uses previously created ISN lists (identified by their command IDs).

Select the ISNs present in both ISN lists identified by the command IDs "CID1" and "CID2".

Search Buffer	`(CID1),D,(CID2).`	Search argument
Value Buffer	not used	Field value specification

Select the ISNs of all the records in file 1 for which an ISN is present in the ISN list identified by "CID1" and which contain the value "+123" for the field AB.

Search Buffer	(CID1),D,AB,3,U.	Search argument

Value Buffer	character-notation
	`123`
	hex-notation
	`F1F2F3`
		Field value specification

Example 12: Using a Value Operator

A search in which a value operator is used.

Select the ISNs of all the records in file 1 that contain a value greater than "+100" for the field AB.

Search Buffer	**`AB,3,U,GT.`**	Search argument

Value Buffer	character-notation
	`100`
	hex-notation
	`F1F0F0`
		Field value specification

Example 13: Using Both Value and Connecting Operators

A search in which both value and connecting operators are used.

Select the ISNs of all the records in file 1 that contain a value greater than "+100" for the field AB and a value greater than "A" for the field AA.

Search Buffer	**`AB,3,U,GT,D,AA,1,GT.`**	Search argument

Value Buffer	character-notation
	`100A`
	hex-notation
	`F1F0F0C1`
		Field value specification

Example 14: Using Physically Coupled Files

A search using search expressions that refer to physically coupled files.

Select the ISNs of all the records in file 1 that contain the value "+20" for the field AB and are coupled to records in file 2 that contain the value "ABCDE" for field RB. Length and format override are used for field AB.

Search Buffer	`/1/AB,3,U,D,/2/RB.`	Search argument

Value Buffer	character-notation
	`020ABCDEbbbbb`
	hex-notation
	`F0F2F0C1C2C3C4C54040404040`
		Field value specification

Example 15: Using Single Soft Coupling Criterion and Single Search Criterion

The file for which ISNs will be returned is determined by the file number field.

File Number	4	
Search Buffer	`(4,AB,1,AC)/1/AB,S,AB.`	Search argument
Value Buffer	`----------`	Field value specification

1. Search file 1 for AB = value as provided in value buffer.
2. For each resulting ISN in file 1, read field AC and internally match the value with the corresponding value list for file 4. The resulting ISN list from file 4 is provided in the ISN buffer.

Example 16: Using Single Soft Coupling Criterion and Multiple Search Criteria

The file for which ISNs will be returned is determined by the file number field. The order in which the criteria are specified is arbitrary.

File Number	1	
Search Buffer	`(1,AA,2,AB)/1/AC,D,AE,D,/2/AF,S,AF.`	Search argument
Value Buffer	`----------`	Field value specification

1. Search file 2 for AF = ... through ... (values in value buffer)
2. For each resulting ISN in file 2, read field AB and internally match the value with the corresponding value list for file 1.
3. Search file 1 for AC = ... and AE = ... (values in value buffer).
4. Match resulting ISN lists from steps 2 and 3. The resulting ISNs are provided in the ISN buffer.

Example 17: Using Multiple Soft Coupling Criteria and Multiple Search Criteria

File Number	1	
Search Buffer	`(1,AA,2,AB; 1,AA,5,BA)` `/1/AC,D,AE,D,/2/AF,S,AF,D,/5/BC,S,BC`	Search argument
Value Buffer	`----------`	Field value specification

1. Search file 2 for AF = ... through ... (values in value buffer).
2. For each resulting ISN in file 2, read field AB and internally match the value with the corresponding value list for file 1.
3. Search file 5 for BC = ... through ... (values in value buffer).
4. For each resulting ISN in file 5, read field BA and internally match the value with the corresponding value list for file 1.
5. Search file 1 for AC = ... and AE = ... (values in value buffer).
6. Match resulting ISN lists from steps 2, 4 and 5. The resulting ISNs are provided in the ISN buffer.

Example 18: Using Multiple Soft Coupling Criteria and Multiple Search Criteria with Physical Coupling

Search Buffer	`(1,AA,2,AB)` `/1/AC,D,AE,D,/2/AF,S,AF,D,/5/BC,S,BC`	Search argument
Value Buffer	`----------`	Field value specification

1. Search file 2 for AF = ... through ... (values in value buffer)
2. For each resulting ISN in file 2, read field AB and internally match the value with the corresponding value list for file 1.
3. Search file 5 for BC = ... through ... (values in value buffer)
4. For each resulting ISN in file 5, use the physical coupling lists created by the ADAINV utility to perform ISN matching. This is done since no soft coupling criteria was provided from file 5 to the main file (file 1). If a physical coupling list does not exist, any ISNs from file 5 will not be considered.
5. Search file 1 for AC = ... and AE = ... (values in value buffer).
6. Match resulting ISN lists from steps 2, 4 and 5. The resulting ISNs are provided in the ISN buffer.

ISN Buffer

Adabas returns the ISNs of the records that satisfy the search criteria in the ISN buffer.

The four-byte binary ISNs are provided in ascending sequence. For the S2 or S9 command, the ISNs are provided according to the user-specified sort sequence. If the ISN buffer is not large enough to contain the entire resulting ISN list, Adabas stores the overflow ISNs on the Adabas Work dataset, if requested. These overflow ISNs may then be retrieved at a later time.

If the resulting ISNs are to be read using the GET NEXT option of the L1/L4 command, the ISN buffer is not needed.

The ISN buffer also supplies an ISN list when the ET or BT command specifies the hold ISN (P) option.

If the prefetch feature is invoked with a command option rather than with ADARUN, the ISN buffer also holds prefetched records awaiting processing.

If the multifetch option is used, Adabas returns prefetched records in the record buffer and corresponding record descriptor elements in the ISN buffer. When used with ET/BT commands, the multifetch option releases only the subset of the records held by the current transaction that are specified in the ISN buffer; see the section *Multifetch Operation Processing* for more information.